**Parcial 1 IS**

**Tema 1: Introducción a la Ingeniería del Software.**

**Introducción**

**Software:** Todo aquello intangible (no físico) que hay en un ordenador, incluyendo el conjunto de programas y el resto de los datos que este ordenador manipula y almacena.

**Hardware:** Conjunto de componentes físicos de un ordenador.

**Desarrollo de software:** Acto de producir o crear software. Conjunto de actividades que nos llevan desde una determinada idea sobre lo que queremos hasta el resultado final del software. Actividades del desarrollo: Programación, compilación, estudio y documentación de las necesidades, mantenimiento…

**Peculiaridades del software:**

* El software es conceptual.
* No tiene propiedades físicas y no está sujeto a leyes físicas o eléctricas.
* Distancia intelectual entre el software y el problema que el software resuelve.
* Dificultad para entender el problema y el sistema software que lo resuelve.
* Para probar es necesario disponer de un sistema físico.
* El mantenimiento no es solo una sustitución de componentes.

Desarrollar software de calidad con el mínimo coste posible es una actividad muy compleja.

**Ingeniería de Software en inicio de la informática:** Se crearon grandes ordenadores dedicados a cálculos matemáticos (computadoras).

**Ingeniería de Software en la actualidad:** Hay una tendencia de dispositivos electrónicos cada vez más pequeños y con más potencia.

**Ámbitos del software**, que puede pertenecer a más de una categoría:

* **Software de sistemas:** Dan servicio a otros programas. No tienen un propósito específico. Por ejemplo: un servidor de bases de datos. Estos programas suelen interactuar directamente con el hardware, sus usuarios no son los usuarios finales que usan el ordenador, sino otros programadores.
* **Software de aplicación:** Programas independientes que resuelven una necesidad específica, normalmente de una organización. Por ejemplo, sistema de citas de un centro de salud. Pueden ser desarrollados a medida o como software de propósito general.
* **Software científico y de ingeniería:** Enfocados al cálculo y a la simulación. Caracterizados por la utilización de algoritmos y modelos matemáticos complejos.
* **Software empotrado:** Forma parte de algún dispositivo. Ejemplos: sistemas de frenado, lavadoras...
* **Aplicaciones web:** Unifican fuentes de datos y diferentes servicios en entornos altamente distribuidos. Las metodologías agiles son muy populares.
* **Software de inteligencia artificial (IA):** Programa que realiza tareas que requieren inteligencia humana, como el aprendizaje o la toma de decisiones. Utiliza algoritmos como el aprendizaje automático para su mejora de rendimiento.

En general, las técnicas de ingeniería del software están **enfocadas** al desarrollo de software de aplicación, y en concreto **a sistemas de información**.

**Sistemas de información:** Conjunto de elementos orientados al tratamiento y administración de datos e información, organizados y listos para cubrir una necesidad u objetivo. El software para sistemas de información gestiona una información mediante un sistema gestor de bases de datos y soporta actividades humanas dentro del contexto de un sistema de información.

**Evolución Histórica**

**Años 60-70:**

* **Pocas computadoras**, grandes y muy caras.
* Poca variedad de software, muy especifico y desarrollado por el fabricante. Difícil programar por libre
* **Se desarrolla software artesanal**. El negocio se encuentra en el hardware, poca atención al software.
* Se disponía del código fuente y los desarrolladores de software compartían sus programas con ánimo constructivo (en los 80 un detonante del movimiento de **software libre**).
* No se planificaba el desarrollo de software, era prácticamente imposible predecir sus costes y tiempo de desarrollo. Se aplicaban algunas técnicas de reutilización, como programación modular.

El software era un producto gratuito que incluía el hardware y era desarrollado por las compañías fabricantes del hardware. Pocas compañías desarrollaban software a medida, pero no existía el software empaquetado como producto.

En los primeros años estaban enfocados en desarrollar el hardware y reducir los costes de procesamiento y almacenamiento. 1962 primer algoritmo para búsquedas binarias.

* 1962, primer algoritmo para búsquedas binarias.
* 1966 fundación para la eliminación de “GoTo” y la creación de la programación estructurada.
* 1968 debe entre el uso de GoTo, y la programación estructurada.
* 1974 primera publicación sobre programación estructurada.
* 1976 primer libro sobre métrica de software.
* 1976 primer libro sobre análisis de requisitos.

**Años 80:**

* Desarrollo de microelectrónica.
* Más potencia de cálculo y menos costes.
* Primeros lenguajes con programación estructurada.
* Empieza la importancia en el diseño (herramientas CASE) y los costes.
* Inicio del movimiento de **software libre**. Movimiento social y filosófico que promueve la libertad de los usuarios a estudiar software. Asumiendo que es un bien público, accesible y controlado por usuarios y no por licencias.

**Aumento de los problemas** de desarrollo **de software** por la explotación del hardware e incapacidad de mantener el software existente.

**Años 90:**

* Imposición del desarrollo orientado a objetos (nace Java).
* Crece la atención a la arquitectura del software (componentes).
* Crece el uso de arquitecturas distribuidas (cliente/servidor).
* Primeros sistemas de control de versiones (CVS).

**Años 2000:**

* Aparecen los entornos de desarrollo integrados (IDEs).
* Se impone UML y decaen las herramientas CASE.
* Nuevos lenguajes (C#, Scala, Go…).

**Años 2010:**

* Auge de sistemas web y software como servicio.
* Metodologías agiles de desarrollo.
* Irrumpe la industria móvil.
* Crecimiento de sistemas empresariales y gubernamentales.

**Actualidad:**

* Heterogeneidad de sistemas: webs, apps…
* Volatilidad de las tecnologías (Más competencia).
* Importancia de la reutilización.
* Evolución constante de los lenguajes.
* Escalabilidad y disponibilidad global.
* Herramientas de soporte y automatización de fases:
  + Frameworks de desarrollo.
  + Desarrollo de CI/CD (Continuous integration / Continuous deployment).
  + Automatización de pruebas (Cypress).
  + Asistentes inteligentes de programación (Copilot).

**Naturaleza y Problemas del Desarrollo de Software**

El software desarrolla en lugar de fabricarse, su creación implica un proceso de diseño y programación, no una producción. La calidad del software depende del diseño y no de la materia prima, siendo los costos más importantes el desarrollo y mantenimiento. El software, con el tiempo, tiende a “deteriorarse” debido a cambios y actualizaciones que pueden introducir errores o complejidad. Al inicio de su historia, el software se desarrollaba a medida, no existían componentes reutilizables ni herramientas de desarrollo estandarizadas.

**La crisis del software**

En 1968, la OTAN (Organización del Tratado del Atlántico Norte) desarrollo la primera conferencia sobre desarrollo de software, donde se acuño “crisis del software”, problemas que surgían en el desarrollo de sistemas software e “ingeniería del software”, conjunto de conocimientos que existían en aquel estado inicial.

La crisis del software se refiere a la dificultad de escribir programas sin defectos, comprensibles y verificables.

Síntomas de crisis en el desarrollo de software:

* Baja **productividad** en relación con la demanda.
* Los sistemas no llegan a las **expectativas**.
* Los programas no son **fiables**.
* **Calidad** no adecuada.
* **Costes** difíciles de predecir, suelen sobrepasar lo esperado.
* Modificación y **mantenimiento** del software costoso y complejo.
* No se cumplen los **plazos**.
* Difícil **portabilidad**.
* No se aprovechan **eficientemente** los recursos.

La productividad y calidad del software disminuyeron. Como solución, se propuso aplicar principios de ingeniería en la construcción de sistemas informáticos.

**Curva de hardware y software**

* Diagrama

  Descripción generada automáticamenteFigura A, **“curva en U” del hardware:** Alta tasa de fallos al principio por defectos de diseño. Disminuye a un nivel estable y bajo una vez corregidos los defectos iniciales. Con el tiempo aumenta por el desgaste acumulativo. El hardware se desgasta con el tiempo.
* Figura B, **“curva idealizada” del software:** Alta tasa de fallos al principio por defectos ocultos que se corrigen, estabilizándose a un nivel bajo. La **“curva real” del software** muestra que la tasa puede aumentar cada vez que se realizan cambios o actualizaciones. El software no se desgasta físicamente, se deteriora debido a la introducción de nuevos errores. No hay refacciones para el software, su mantenimiento es más complejo.

**Mitos del software**

Mitos de los gestores:

* **Uso de estándares de otras ingenierías:** Creen que las metodologías y estándares de otras disciplinas de ingeniería son aplicables al desarrollo de software. Sin embargo, el software es un elemento lógico, no físico, y su desarrollo sigue ciclos más flexibles e iterativos, por lo que necesita métodos específicos.
* **Uso de herramientas:** Creen que el uso de herramientas de software garantiza el éxito del proyecto. Sin embargo, las herramientas son solo tan eficaces como el equipo que las utiliza y deben estar integradas adecuadamente.
* **Mala planificación:** Creen que añadir más programadores a un proyecto acelerará su finalización, esto suele generar retrasos adicionales debido al tiempo integración de los nuevos miembros y coordinar su trabajo con el equipo existente.

Mitos de los desarrolladores:

* **Programa funcionando = fin del trabajo;** Sin embargo, el software necesita pruebas exhaustivas, mantenimiento, optimización y documentación. Su desarrollo continúa incluso después de que el programa es funcional.
* **Calidad = el programa se ejecuta sin errores;** Sin embargo, la calidad del software incluye aspectos como la facilidad de uso, la seguridad, la escalabilidad y la capacidad de ser mantenido y modificado eficientemente.
* **Entrega al cliente = programa funcionando;** La entrega es solo una fase; se requiere capacitación, pruebas adicionales, ajustes, y soporte posterior para asegurar que el software cumple las expectativas del cliente.

Mitos del cliente:

* **Requisitos establecidos como una declaración general de objetivos:** Sin embargo, los requisitos deben ser específicos y detallados para evitar malentendidos y garantizar que el software cumpla sus necesidades.
* **Flexibilidad del software ante los cambios:** Los clientes piensan que el software puede ser modificado fácilmente en cualquier etapa del proyecto. Aunque el software es flexible, los cambios significativos en fases avanzadas pueden ser costosos y difíciles de implementar sin afectar la calidad y los plazos de entrega.

**La Ingeniería del Software**

**Ingeniería del software:** Establecimiento y uso de principios de ingeniería para obtener software económico que trabaje de forma eficiente en máquinas reales.

**Aspectos esenciales:**

Principios de ingeniería orientados a obtener software fiable, eficiente y que satisfaga las necesidades del usuario. **Disciplina** que comprende **todos los aspectos de la producción de software**:

* “disciplina de ingeniería”: aplicación de teorías, métodos y herramientas para solucionar problemas, y teniendo en cuenta las restricciones.
* “todos los aspectos de producción”: procesos técnicos del desarrollo y actividades como la administración de proyectos, métodos y teorías.

No solo es programar, engloba planificar, modelar... Se orienta a la solución de problemas y adquisición de conocimiento.

Desde 1968 han sido muchos los esfuerzos para identificar las causas del problema y definir pautas estándar para la producción y mantenimiento del software. Se han encaminado en tres direcciones principales:

* Identificación de los factores clave que determinan la calidad del software.
* Identificación de los procesos necesarios para producir y mantener software.
* Acotación, estructuración y desarrollo de la base para la producción y mantenimiento de software.

El resultado ha sido la necesidad de profesionalizar el desarrollo, mantenimiento y operación de los sistemas de software, introduciendo métodos y formas de trabajo sistemáticos, disciplinados y cuantificables.

**Ingeniería del Software:** aplicar el sentido común al desarrollo de sistemas software

**Estandarización de la IS**

Desde la identificación de la “crisis del software” se han ido generando normas y estándares. Se consideran como entidades de mayor reconocimiento internacional:

* **Organización Internacional para la Estandarización (ISO**
* **IEEE Computer Society:** Instituto de Ingenieros en electricidad y electrónica (Institute of Electrical and Electronics Engineers).
* **Instituto de Ingeniero del Software (SEI):** Integrado en la Universidad Carnegie Mellon.

**Estándares y necesidades de la Ingeniería del Software**

* **Definirse a sí misma:** SWEBOK (Software Engineering Body of knowledge): Áreas de conocimiento que comprenden la IS.
* **Definir los procesos que intervienen en el desarrollo, mantenimiento**
* **y operación del software**: ISO/IEC 12207: Procesos del ciclo de vida del software.
* **De las mejores prácticas, extraer modelos de cómo ejecutar esos**

**procesos para evitar problemas:** CMM/CMMI, ISO/IEC TR 15504.

* **Definir estándares menores para dibujar criterios unificadores**: IEEE 830-IEEE 1362, ISO/IEC 14764, UML...

**Conceptos de la Ingeniería del Software**

**Rol:** Conjunto de responsabilidades dentro del proyecto o sistema. Está vinculado a un conjunto de tareas y se asigna a un participante. Roles en IS:

* **Cliente:** encarga y paga el sistema.
* **Equipo de desarrollo:** construyen el sistema.
* **Gerente o director del proyecto:** planifica y calcula el presupuesto, coordina a desarrolladores y cliente.
* **Usuarios finales:** los que van a utilizar el sistema.

**Ingeniero de software:** Se encarga de diseñar, desarrollar, probar y mantener aplicaciones o sistemas de software. Trabaja en todas las etapas del software, desde la planificación hasta la implementación y el mantenimiento. Su trabajo incluye escribir código, depurar programas, optimizar el rendimiento y colaborar con otros equipos para cumplir con las especificaciones del cliente.

**Habilidades necesarias para un ingeniero de software:** Un ingeniero de software necesita comprender lenguaje de programación, algoritmos y estructuras de datos, así como experiencia con bases de datos y metodologías de desarrollo. Debe saber usar herramientas de control de versiones y pruebas de software, además de tener habilidades en diseño y arquitectura de software. Debe ser capaz de adaptarse a nuevas tecnologías y cambios en el proyecto.

**Sistema:** realidad subyacente.

**Modelo:** cualquier abstracción de la realidad.

**Producto o entregable:** Elemento que se produce durante el desarrollo.

* Producto de trabajo interno: producto para el consumo interno del proyecto.
* Entrega: producto de trabajo para un cliente.

**Actividades o fases:** Conjunto de tareas que se realiza con un propósito específico y que pueden componerse de otras actividades.

**Tareas:** Unidad elemental de trabajo que puede ser administrada, consumen recursos, dan como resultado productos de trabajo.

**Recursos:** Bienes que se utilizan para realizar el trabajo. Al planificar, el gerente divide el trabajo en tareas y les asigna recursos.

**Objetivos:** Principios que se utilizan para guiar el proyecto. Definen los atributos importantes del sistema.

**Requerimientos:** Características que debe tener el sistema.

* **Requerimiento funcional:** Área de funcionalidad que debe soportar el sistema.
* **Requerimiento no funcional:** Restricción que se establece sobre el funcionamiento del sistema.

**Notaciones:** Conjunto de reglas gráficas o de texto para representar un modelo.

**Métodos:** Técnica repetible para resolver un problema específico.

**Metodologías:** Colección de métodos para la resolución de una clase de problemas.

**Principios de la Ingeniería del Software**

* Haz de la calidad la razón de trabajar.
* Una buena gestión es más importante que una buena tecnología.
* Las personas y el tiempo no son intercambiables.
* Seleccionar el modelo de ciclo de vida adecuado.
* Determinar y acotar el problema antes de escribir los requisitos.
* Minimizar la distancia intelectual.
* Las técnicas son anteriores a las herramientas.
* Primero hazlo correcto, luego hazlo rápido.
* Introducir las mejoras y modificaciones con cuidado.
* Asume responsabilidades.
* La entropía del software es creciente.
* La gente es la clave del éxito.
* Nunca dejes que tu jefe o cliente te convenza para hacer mal un trabajo.
* La gente necesita sentir que su trabajo es apreciado.
* La educación continua es responsabilidad de cada miembro del equipo.
* El compromiso del cliente es el factor más crítico en la calidad del software.
* Tu mejor desafío es compartir la visión del producto con el cliente.
* La mejora continua de tu proceso de desarrollo de software es posible y esencial.
* Tener procedimientos escritos de desarrollo de software puede ayudar a crear una cultura compartida de buenas prácticas.
* La calidad es el principal objetivo; la productividad a largo plazo es una consecuencia de alta calidad.
* Haz que los errores los encuentre un colaborador y no un cliente.
* Una clave en la calidad en el desarrollo de software es realizar iteraciones en todas las fases de desarrollo.
* La gestión de errores y solicitud de cambios es esencial para controlar calidad y el mantenimiento.
* Si mides lo que haces, puedes aprender a hacerlo mejor.
* Haz lo que tenga sentido, no recurras a los dogmas.
* No se puede cambiar todo de una vez.
* Identifica los cambios que se traduzcan en los mayores beneficios, y comienza a implementarlos.

**Tema 2: El proceso de desarrollo del software.**

**El Proceso: Modelos de Desarrollo**

En software, un **proceso** es un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema funcional.

**Características:**

* Tiene una serie de actividades principales.
* Utiliza recursos, está sujeto a restricciones y genera productos intermedios y finales.
* Compuesto por subprocesos que se encadenan de alguna forma.
* Cada actividad tiene sus criterios de entrada y salida, que permiten conocer cuando comienza y termina dicha actividad.
* Existen principios orientadores que explican las metas de cada actividad.

Según el estándar ISO/IEC 1220:

* **Ciclo de vida del software:** El periodo de tiempo comprendido desde la definición de los requisitos hasta el fin del su uso.
* **Procesos:** Conjunto de actividades y tareas interrelacionadas que juntas transforman entrada en salida.

**Procesos primarios del ciclo de vida del software**

* **Adquisición:** Proceso global que sigue el adquiriente para obtener el producto.
* **Suministro:** Proceso global que sigue el suministrador para proporcionar el producto.
* **Desarrollo:** Proceso empleado por el suministrador para el diseño, construcción del producto.
* **Operación:** Proceso que sigue el operador en su uso del producto.
* **Mantenimiento:** Proceso destinado a mantener el producto.

**Procesos de soporte del ciclo de vida del software**

El estándar 12207 identifica los procesos de soporte que pueden utilizarse desde un proceso primario como en otro proceso de soporte.

* **Documentación:** Actividades empleadas para registrar información específica empleada por otros procesos.
* **Gestión de la configuración:** Actividades empleadas para mantener un registro de los productos generados en la ejecución de los procesos.
* **Aseguramiento de la calidad:** Actividades empleadas para garantizar de forma objetiva que el producto y los procesos asociados son conformes a los requisitos documentados y a las planificaciones.
* **Verificación:** Actividades empleadas para comprobar el producto.
* **Validación:** Actividades empleadas para confirmar que el producto cumple su propósito.
* **Reuniones de revisión:** Reuniones empleadas por las dos partes para evaluar el estado del producto y de las actividades.
* **Auditorias:** Actividades para determinar que el proyecto cumple con los requisitos, planes y contratos.
* **Resolución de problemas:** Actividades para analizar y resolver problemas relativas al proyecto, sea cual sea su fuente y naturaleza.

**Procesos organizacionales**

El estándar 12207 identifica los procesos que deben llevarse a cabo dentro de la organización encargada de ejecutar el proyecto. Generalmente se aplican de manera común a múltiples proyectos.

* **Gestión:** Describe las actividades de gestión de la organización, incluyendo la de proyectos.
* **Infraestructura:** Actividades necesarias para que puedan realizarse otros procesos del ciclo de vida.
* **Mejora:** Actividades realizadas para mejorar la capacidad del resto de procesos.
* **Formación:** Actividades para capacitar y desarrollar al personal.

**Proceso de desarrollo de software**

El proceso de desarrollo de software se divide en actividades, las cuales se descomponen en acciones y tareas. Se pueden identificar cinco actividades principales: comunicación, planificación, modelado, construcción y despliegue. Además, existen actividades “paraguas” que incluyen la gestión y control del proyecto, la gestión del riesgo, el aseguramiento de la calidad y la gestión de la configuración, entre otras.

En las tareas intervienen:

* Productos o artefactos que se utilizan (entrada) o construyen (salida).
* Aspectos de control de la calidad.
* Hitos de proyecto.

Cada actividad estructural está formada por un conjunto de acciones de ingeniería de software y estas se encuentra definidas por un conjunto de tareas que identifica que debe realizarse, los productos del trabajo que se producirán, de aseguramiento de la calidad que se requieren y de referencia que se utilizaran.

En la foto se muestra este aspecto, conocido como **flujo del proceso**, que describe cómo están organizadas las actividades estructurales y las acciones y tareas que ocurren dentro de cada una en relación con la secuencia y el tiempo. Los tipos de flujos son los siguientes:

* **Flujo de proceso lineal:** ejecuta las cinco actividades estructurales en secuencia, desde comunicación a despliegue.
* **Flujo de proceso iterativo:** repite una o más de las actividades antes de pasar a la siguiente.
* **Flujo de proceso evolutivo:** realiza las actividades en forma “circular”. A través de las cinco actividades, cada circuito lleva a una versión más completa del software.
* **Flujo de proceso paralelo:** ejecuta una o más actividades en paralelo con otras.

**Paradigmas o Modelos de Desarrollo del Software**

Son un conjunto de técnicas y métodos organizativos diseñados para optimizar la creación de soluciones informáticas. Su objetivo es estructurar los equipos de trabajo para ser eficientes.

Al desarrollar productos es crucial considerar factores como costes, planificación y complejidad. Estas se integran en una metodología que organiza el trabajo de forma ordenada.

El desarrollo de software puede ser complejo en proyectos grandes. Su desarrollo sin una metodología clara puede llevar a problemas, retrasos y errores, resultando un producto deficiente. Utilizar una metodología adecuada ayuda a reducir la complejidad, organizar las tareas y mejorar la calidad del resultado final.

Existen dos grandes grupos de metodologías de desarrollo de software:

* **Metodologías de desarrollo de software tradicionales:** Se caracterizan por definir de manera rígida los requisitos al inicio de los proyectos, haciéndolos inflexibles ante cambios. Su enfoque implica que las etapas deben completarse sin posibilidad de retroceso, lo que dificulta la adaptación en un entorno en constante cambio.
* **Metodologías de desarrollo de software agiles:** Tiene un enfoque incremental, donde cada ciclo agrega nuevas funcionalidades de forma rápida y en pequeñas porciones. Fomentan la formación de equipos autosuficientes que se reúnen regularmente para compartir novedades, permite al cliente aporte requerimientos y correcciones a medida que el proyecto avanza, asegurando un desarrollo colaborativo y en tiempo real.

**Modelos Tradicionales de Desarrollo de Software**

**Lineal o secuencial**

Se caracteriza por la sucesión escalonada de etapas: requisitos, diseño, codificación, pruebas e integración. Se debe completar cada etapa antes de pasar a la siguiente, haciéndolo muy rígido. Este modelo es problemático, ya que cada fase requiere como entrada el resultado completo de la anterior. Pudiendo generar dificultades, es complicado contar con requisitos completos o un diseño detallado en fases iniciales. Este modelo puede ser adecuado en ciertas circunstancias, como:

* **Desarrollo de nuevas versiones de sistemas ya establecidos**, donde el desconocimiento de las necesidades del usuario y del entorno operativo no plantea riesgos significativos.
* **Sistemas pequeños**, sin previsión de evolución a corto plazo.

**Cascada**

Este modelo refleja la necesidad de regresar con frecuencia desde una fase hacia las anteriores, utilizando la información generada a medida que avanza el desarrollo. El retorno puede ocurrir únicamente entre una fase y la anterior, es decir, en cualquier fase puede haber necesidad de modificar las etapas previas. Da importancia a tener requisitos y un diseño bien definidos antes de comenzar la codificación. Este modelo es problemático, ya que cada fase requiere como entrada el resultado completo de la anterior. Pudiendo generar dificultades, es complicado contar con requisitos completos o un diseño detallado en fases iniciales. Este modelo puede ser adecuado en ciertas circunstancias, como:

* **Desarrollo de nuevas versiones de sistemas ya establecidos**, donde el desconocimiento de las necesidades del usuario y del entorno operativo no plantea riesgos significativos.
* **Sistemas pequeños**, sin previsión de evolución a corto plazo.

Es importante señalar que algunos textos utilizan el término **“cascada”** para referirse al **modelo lineal**, y **“cascada modificada”** para describir el **modelo en cascada**.

**Espiral**

Este modelo presenta un enfoque evolutivo al desarrollo de software. Introduce el “análisis de riesgo”, que guía la evolución del proceso de desarrollo.

El ciclo de **iteración** de este modelo se convierte en una espiral, que al representarse sobre ejes cartesianos muestra en cada cuadrante una clase particular de actividad: Planificación, Análisis de riesgo, Ingeniería y Evaluación, que se suceden consecutivamente a lo largo del desarrollo. La dimensión angular representa el avance relativo en el desarrollo de las actividades de cada cuadrante. En cada ciclo de la espiral se realiza una parte del desarrollo, a través de los cuatro tipos de actividades.

* En la **planificación** de cada vuelta se establece el contexto del desarrollo y qué parte se abordará en el ciclo siguiente.
* Las **actividades de análisis de riesgo** evalúan las alternativas para la siguiente parte del desarrollo, seleccionando la más ventajosa y previendo los riesgos.
* Las actividades de **ingeniería** corresponden a las indicadas en los modelos lineales (secuencial y cascada): análisis, diseño, codificación...
* Las actividades de evaluación analizan los resultados de la fase de ingeniería, tomando el resultado de **evaluación** como punto de partida de la siguiente fase.

Permite múltiples combinaciones, en la planificación de cada ciclo se determina el avance que se va a ejecutar en la vuelta. Puede consistir en obtención y validación de requisitos, o en desarrollo del diseño, o en diseño de codificación, o en obtención de un subsistema (cascada de requisitos – diseño – codificación – pruebas – integración).

**Incremental**

Elimina la rigidez del modelo en cascada, descomponiendo el desarrollo de un sistema en partes donde a cada una se le aplica un ciclo de desarrollo. Ventajas:

* El usuario dispone de pequeños subsistemas operativos que obtener las necesidades reales del sistema en su conjunto.
* El modelo produce entregas parciales en periodos cortos y permite la incorporación de nuevos requisitos que pueden no estar disponibles o no ser conocidos al iniciar el desarrollo.

Los desarrollos de cada subsistema pueden solaparse en el tiempo o ser secuenciales. A la funcionalidad seleccionada para el primer incremento se le conoce como producto “núcleo” (core product). Este enfoque resulta apropiado para:

* Desarrollo de sistemas en los que el cliente necesita disponer de parte de la funcionalidad antes de desarrollar el sistema completo.
* Desarrollo de sistemas en los que interesa realizar la obtención de los requisitos de forma escalonada a través de subsistemas.

**Evolutivo**

Este modelo se compone de varios ciclos de desarrollo, cada ciclo produce un sistema completo que se usa en el entorno operativo. La información acumulada en cada periodo se utiliza para mejorar y ampliar los requisitos y el diseño del siguiente. Representa un ciclo de vida aplicable a todos los sistemas desarrollados, que se optimizan a través de versiones sucesivas.

Las circunstancias en las que este modelo puede resultar apropiado son:

* Desconocimiento inicial de todas las necesidades operativas que serán precisas, por desarrollarse en un entorno nuevo sin experiencia previa.
* Necesidad de que el sistema entre en operación en tiempos inferiores a los necesarios para diseñarlo de forma exhaustiva.
* Necesidad de desarrollar sistemas en entornos cambiantes.

Aunque en su concepción inicial contempla desarrollos internos en cascada, también podría plantearse, por ejemplo, en espiral.

**Prototipado**

Se basa en la construcción de un prototipo de software rápido para que los usuarios puedan aportar feedback. Después, arreglar lo que está mal e incluir otros requerimientos que surgan. Se basa en el método de prueba y error.

Los prototipos pueden ser:

* **Ligeros:** Dibujos de pantallas de interfaz con simulación de funcionamiento por enlaces a otros dibujos.
* **Operativos:** Módulos de software con funcionamiento propio que se desarrollan sin cubrir las funcionalidades completas; entornos RAD (Diseño rápido de aplicaciones).

Esta forma de trabajo tiene como objetivo la experimentación con un entorno similar al pretendido, para retroinformarse del usuario o cliente y concretar requisitos. Desventajas:

* Puede parecer que se ha desarrollado un interfaz de usuario elaborada y hacer creer al cliente que ya se ha realizado el grueso del trabajo.
* Es un prototipo operativo, puede empezar a crecer al margen de la planificación, más allá de previsto, desbordando agendas y recursos.
* Si se trata de un prototipo ligero desarrollado fuera del departamento de desarrollo, puede mostrar al cliente funcionalidades no implementables.
* El prototipo puede llegar a ofrecer funcionalidades superiores o inferiores a lo conseguible al estar construido en un entorno distinto al de desarrollo.

**Concurrencia**

Consiste en el solapamiento de actividades en un estado concreto. Puede aportar beneficios o ser origen o consecuencia de problemas. Factores a tener en cuenta:

* Índice de concurrencia: Se produce en un grado reducido, generando un escaso flujo de modificaciones o intensivo, generando problemáticas en la planificación o distribución del trabajo.
* Gestión de la concurrencia: Puede producirse en un proyecto de forma planificada o inducida por las circunstancias. En ambos casos es muy importante la gestión del proyecto para tratarla de forma adecuada.

**Proceso Unificado de Desarrollo**

El Proceso Unificado de Desarrollo de Software es un marco de desarrollo de software caracterizado por estar dirigido por casos de uso, centrado en arquitectura y por ser iterativo e incremental. Su objetivo era ser para los métodos de desarrollo de software lo que UML para los lenguajes de modelización. Integra elementos de los enfoques tradicionales y agiles, por lo que se sitúa entre de ambos.

El Proceso Unificado (UP) consiste en un conjunto de procesos configurables que pueden adaptarse a diferentes contextos, lo que significa que su aplicación varía según el tipo de desarrollo, ya sea una intranet empresarial o un software militar. Facilita el desarrollo en ciclos cortos, permitiendo una adaptación continua. Si bien es más rígido que metodologías ágiles puras como Scrum o XP, ofrece retroalimentación y acepta cambios durante el desarrollo, haciéndolo adaptable a diversas situaciones del proyecto.

En términos generales todas las variantes del proceso unificado se basan en potenciar seis prácticas fundamentales:

1. Desarrollo iterativo e incremental.
2. Gestión de los requisitos (mediante casos de uso).
3. Arquitecturas basadas en componentes (módulos).
4. Utilización de modelos visuales (con UML).
5. Verificación de la calidad del software (en todas las etapas).
6. Control de los cambios al software.

El proceso se puede descomponer de dos formas: por tiempo, organizado en fases o en función de las actividades o contenido, desglosándolo en tareas específicas. Definiendo los roles de las distintas personas involucradas y las tareas que debe realizar cada uno.

**Fases del proyecto**

El UP se repite a lo largo de una serie de ciclos en el tiempo de manera secuencial y consta de cuatro fases:

1. **Inicio:** Se obtiene idea inicial del producto y se realiza un análisis de negocio, identificando las principales funciones del sistema y los usuarios más relevantes mediante casos de uso. También se elabora un plan del proyecto que incluye costos, identificación y priorización de riesgos.
2. **Elaboración:** Se detallan los casos de uso y diseña la arquitectura del sistema, incluyendo las vistas arquitectónicas del modelo de casos, análisis, diseño, implementación y despliegue. Al finalizar, se planifican las actividades y se estiman los recursos necesarios para completar el proyecto.
3. **Construcción:** Se desarrolla el producto al integrar el software en la arquitectura definida. Al finalizar, se tienen todos los casos de uso acordados para el desarrollo, aunque pueden presentar defectos.
4. **Transición:** Se centra en la conversión del producto en una versión beta, donde los usuarios prueban el sistema e informan de los defectos. Se corrigen problemas e incorporan sugerencias. Además, incluye formación de los usuarios, la provisión de una línea de ayuda y asistencia.

Cada fase se divide a su vez en iteraciones.

**Modelos Agiles de Desarrollo de Software**

Los modelos de procesos ágiles se basan en el “Manifiesto Ágil de Software”, que fue creado en 2001 por desarrolladores centrados en cuatro valores:

* Individuos e interacciones sobre procesos y herramientas.
* Software funcionando sobre documentación extensiva.
* Colaboración con el cliente sobre negociación contractual.
* Respuesta ante el cambio sobre seguir un plan.

Aunque valoramos los elementos de la derecha, valoramos más los de la izquierda.

El manifiesto Ágil está basado en 12 principios, que son:

1. Nuestra mayor prioridad es satisfacer al cliente mediante la entrega temprana y continua de software con valor.
2. Aceptamos que los requisitos cambien, incluso en etapas tardías del desarrollo. Los procesos Agiles aprovechan el cambio para proporcionar ventaja competitiva al cliente.
3. Entregamos software funcional frecuentemente, entre dos semanas y dos meses, con preferencia al periodo de tiempo más corto posible.
4. Los responsables de negocio y los desarrolladores trabajamos juntos de forma cotidiana durante todo el proyecto.
5. Los proyectos se desarrollan en torno a individuos motivados. Hay que darles el entorno y el apoyo que necesitan, y confiarles la ejecución del trabajo.
6. El método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros es la conversación cara a cara.
7. El software funcionando es la medida principal de progreso.
8. Los procesos Agiles promueven el desarrollo sostenible. Los promotores, desarrolladores y usuarios debemos ser capaces de mantener un ritmo constante de forma indefinida.
9. La atención continua a la excelencia técnica y al buen diseño mejora la Agilidad.
10. La simplicidad, o el arte de maximizar la cantidad de trabajo no realizado, es esencial.
11. Las mejores arquitecturas, requisitos y diseños emergen de equipos autoorganizados.
12. A intervalos regulares el equipo reflexiona sobre cómo ser más efectivo para a continuación ajustar y perfeccionar su comportamiento en consecuencia.

El proceso ágil se puede aplicar cuando:

* Cuando resulta difícil predecir que requisitos del software persistirán y cuales cambiarán (cambios en las prioridades).
* El diseño y la construcción de software deben de estar intercalados.
* El análisis, diseño y construcción de software no son predecibles respecto a la planificación.

La Ingeniería del Software Ágil combina una filosofía y directrices de desarrollo, como:

* Busca la satisfacción del cliente y la entrega temprana del software incremental
* Equipos de proyectos pequeños y con alta motivación
* Métodos informales
* Mismos productos de trabajo
* Simplicidad general del desarrollo

Actualmente, la Ingeniería del Software Ágil es especialmente relevante debido a la aceleración y cambio de los sistemas que incluyen computadoras. Es una alternativa eficaz a la ingeniería tradicional para determinadas clases de software y tipos de proyectos.

**Scrum**

Su proviene de una jugada del rugby, es un método ágil de desarrollo de software creado en los 90. Los principios de Scrum están alineados con el Manifiesto Ágil y guían las actividades de desarrollo dentro de un proceso de análisis que incluye las siguientes fases: requerimientos, análisis, diseño, evolución y entrega.

En cada fase, las tareas se organizan en un patrón de trabajo llamado *sprint*. El trabajo realizado en cada *sprint* (el número de *sprints* varía según el proyecto) se ajusta a las necesidades del problema, es definido y frecuentemente modificado en tiempo real por el equipo Scrum.

**Roles**

* **Scrum Master:** Responsable de asegurar que el equipo sigue las reglas de Scrum. También se encarga de eliminar los impedimentos en organización.
* **Product owner:** Es el encargado de decidir qué se implementa y qué es más prioritario.
* **Team:** El resto de los miembros del equipo son los desarrolladores. Deciden cómo se organizan el trabajo y nadie les puede decir cómo lo deben hacer.

**Artefactos**

* **Product backlog:** Lista de requisitos pendientes de implementar al producto. Cada entrada tiene asociada una estimación del valor que aporta a la organización y de su coste de desarrollo.
* **Sprint backlog:** El backlog para una iteración concreta (En Scrum se llama sprint a las iteraciones); más detallada que el product backlog. Tienen una duración máxima de 6 semanas.
* **Release burndown chart:** Gráfico que muestra el progreso actual del equipo en función del número de historias de usuario que faltan por implementar.
* **Sprint burndown:** El burndown para una iteración concreta, en la que el progreso se puede medir en tareas finalizadas aunque no sean historias de usuario completas.

**Reuniones**

En cuanto a las reuniones, estas se basan en dos ciclos de iteraciones: una iteración más larga y una iteración diaria.

* **Sprint planning meeting:** Reunión que se organiza antes de empezar un sprint y en la que se deciden qué historias de usuario se implementaran en este sprint. Creando un sprint backlog.
* **Daily scrum:** Reunión diaria en la que todos los miembros del equipo responden a: que hicieron ayer, qué piensan hacer hoy y qué impedimentos se han encontrado que les han impedido avanzar. Así, todos los miembros del equipo están enterados de lo que está haciendo el resto de los miembros y se pueden ayudar.
* **Sprint review meeting:** Al finalizar un sprint se revisa el trabajo realizado y se enseña a quien esté interesado en el resultado (demo).
* **Sprint retrospective:** Sirve para reflexionar sobre lo que haya pasado durante el sprint y para identificar oportunidades de mejora en el proceso de desarrollo.

**Otros modelos agiles**

**El modelo XP (Programación Extrema)**

Es una metodología ágil de desarrollo de software enfocada en mejorar la calidad del software y la capacidad de respuesta ante los cambios en los requisitos del cliente. Promueve la colaboración constante entre equipo y cliente, entregas frecuentes y mejoras continuas, con el objetivo de crear software de alta calidad, eficiente y flexible.

Utiliza un enfoque orientado a objetos y sigue un conjunto de reglas y prácticas organizadas en cuatro actividades:

* **Planeación:** Permite al equipo comprender los requisitos del cliente y adaptarse a sus necesidades mediante una comunicación continua y asegurando que el software cumpla las expectativas y pueda ajustarse a los cambios.
* **Diseño:** Promueve la simplicidad y flexibilidad mediante un diseño evolutivo que se ajusta a las necesidades, evitando complejidad y manteniendo un sistema fácil de modificar.
* **Codificación:** Es el núcleo de XP, enfocada en un código claro y sencillo, con programación en parejas para garantizar calidad y detectar errores de forma rápida, fomentando la colaboración y el aprendizaje mutuo.
* **Pruebas:** Garantizan la calidad del software mediante pruebas automatizadas que validan cada funcionalidad antes de su implementación, asegurando la fiabilidad del sistema a través del desarrollo basado en pruebas (TDD).

**Kanban**

Es un método ágil de gestión de proyectos y de flujo de trabajo. Su principal objetivo es mejorar la eficiencia y la productividad al visualizar las tareas y gestionar el flujo de trabajo de manera continua. No requiere iteraciones predefinidas, en su lugar, se centra en gestionar un flujo constante de trabajo y facilitar la entrega continua de productos. Depende de seis principios básicos:

1. Visualizar el flujo de trabajo mediante el uso de un tablero de Kanban dividido en columnas que representan diferentes etapas del proceso de trabajo. Las tareas del tablero podrían contener historias de usuarios o defectos recién descubiertos con comentarios “por hacer” a “haciendo” y luego “hecho”.
2. Limitar la cantidad de trabajo en progreso (WIP) en cada etapa del flujo. Solo se puede trabajar en un número limitado de tareas a la vez y completar las actuales para comenzar nuevas.
3. Gestionar el flujo de trabajo para mantener una entrega continua.
4. Hacer explicitas las políticas del proceso.
5. Enfocarse en la mejora continua mediante retroalimentación.
6. Realizar los cambios en el proceso de forma colaborativa e involucrar a todos los miembros del equipo y demás partes interesadas según sea necesario.

**Tema 3: Planificación de Sistemas Software.**

**Introducción**

Se encuentra enmarcada dentro de la gestión de proyectos. Esta disciplina es fundamental en la ingeniería del software, ya que aunque una buena gestión no garantiza el éxito del proyecto, una mala gestión conduce al fracaso del mismo.

Los gestores de software tienen el mismo trabajo que los de otras áreas, pero presenta características únicas:

* El producto es intangible.
* No existen procesos de software estándar.
* A menudo los proyectos software grandes son únicos.

Nos centraremos en tres aspectos clave de la gestión: la planificación temporal, la gestión de riesgos y la estimación de los costes.

Un proyecto es un conjunto de etapas, actividades y tareas que tiene como finalidad alcanzar un objetivo, a un plazo relativamente largo. Características:

* Implica un principio y un final.
* Utiliza diversos recursos finitos y un presupuesto
* Tiene actividades únicas y esencialmente no repetitivas.
* Tiene un objetivo.
* Requiere un jefe de proyecto y personal de desarrollo cuyos roles deben definirse y desarrollarse.
* Tiene que planificarse.
* Debe medir su progreso frente al plan.
* Suele coexistir con otros proyectos y competir por los recursos.

**Plan del Proyecto**

El contenido del plan del proyecto es variable en cada proyecto, pero es recomendable incluir los siguientes elementos:

* Un resumen del proyecto que pueda ser comprendido por cualquier persona.
* Debe indicar los productos entregables, para cuando se produzcan, comprobar que se ajustan al plan.
* Una lista de los hitos alcanzables.
* Los procedimientos y estándares a aplicar.
* Una especificación del proceso de revisión que determine quién, cómo y cuándo se puede revisar la planificación del proyecto.
* Un diagrama de descomposición del trabajo (WBS).
* Una lista del personal del proyecto y su asignación en relación al WBS.
* Una red de actividades que muestre las tareas en el tiempo y su relación.
* Los responsables de todas y cada una de las actividades.
* Los presupuestos de esfuerzo, costes, calendarios y plazos de las actividades.

Un proyecto de software debe seguir una estructura bien definida que abarca varios contenidos esenciales:

**Plan de proyecto de software**

* Introducción
  + Alcance y propósito del documento
  + Objetivos del proyecto
    - Objetivos
    - Funciones
    - Aspectos de funcionamiento
    - Restricciones técnicas y de gestión
* Estimaciones del proyecto
  + Datos históricos utilizados para las estimaciones
  + Técnicas de estimación
  + Estimaciones
* Riesgos del proyecto
  + Análisis del riesgo
    - Identificación
    - Estimación del riesgo
    - Evaluación
  + Gestión del riesgo
    - Opciones de aversión al riesgo
    - Procedimientos de supervisión del riesgo
* Agenda
  + Estructura de descomposición de trabajos del proyecto
  + Red de actividades
  + Diagrama de Gantt
  + Tabla de recursos
* Recursos del proyecto
  + Personal
  + Hardware y software
  + Recursos especiales
* Organización del personal
  + Estructura de equipos (si existe)
  + Información de gestión
* Mecanismos de seguimiento y control
* Anexos

**Planificación Temporal**

* **Descomposición del proyecto:** Es fundamental dividir el proyecto en un número manejable de tareas más pequeñas y específicas.
* **Interdependencia**: Se deben determinar las dependencias de cada tarea, ya que algunas tareas no pueden comenzar hasta que otras hayan finalizado.
* **Asignación de tiempo:** Cada tarea debe tener asignado un tiempo específico para su ejecución (fecha de inicio y finalización).
* **Validación del esfuerzo:** A medida que se asigna tiempo a las tareas, el gestor del proyecto debe asegurarse de que los técnicos necesarios estén disponibles en el momento adecuado.
* **Responsabilidades definidas:** Cada tarea que se programe debe asignarse a un miembro específico del proyecto.
* **Resultados definidos:** El resultado de cada tarea deberá estar definido. Los productos resultantes suelen combinarse en entregas mayores para facilitar la evaluación del progreso del proyecto en su conjunto.
* **Hitos definidos:** Todas las tareas o grupos de tareas deben asociarse con algún hito del proyecto. Se considera un hito cuando se ha revisado la calidad de al menos productos y se ha aceptado.

Características comunes de los métodos de planificación:

* Identificar las tareas del proyecto y sus dependencias.
* Establecimiento de estimaciones de tiempo para las tareas.
* Estimación de los recursos necesarios para llevar a cabo cada tarea.
* Calcular los tiempos límite para cada tarea.

Dado que la planificación se basa en estimaciones, es importante que se actualice de forma continua y se obtengan nuevos datos.

Existen diversas representaciones visuales que pueden utilizarse para la planificación: basadas en calendario (diagrama de Gantt) o redes de actividades (diagrama de precedencias).

**Diagrama de Gantt**

El diagrama de Gantt es uno de los métodos más antiguos y más utilizados en la planificación y gestión de proyectos. Su simplicidad y claridad lo convierten en una gran herramienta para visualizar el cronograma de trabajo.

Se representa en forma de cuadro de doble entrada:

* **Eje horizontal:** Representa el tiempo, dividido en días, semanas o meses, dependiendo de la duración del mismo.
* **Eje vertical:** Enumera las tareas del proyecto en orden jerárquico o secuencial.
* **Tareas:** Cada tarea se muestra como un rectángulo colocado a la altura correspondiente en el eje vertical. La longitud de cada rectángulo indica la duración de la tarea, desde su inicio hasta su finalización.

A pesar de su utilidad, el diagrama de Gantt presenta algunos inconvenientes:

* **Falta de interdependencias:** No muestra de manera explícita las relaciones entre tareas, pudiendo dificultar la identificación de dependencias críticas en el proyecto.
* **Complejidad en proyectos grandes:** En proyectos con muchas tareas, el diagrama puede volverse complicado y difícil de crear, pudiendo verse desordenado y confuso.

El diagrama de Gantt también ofrece varias ventajas significativas:

* **Facilidad de comprensión:** Su diseño simple y visual lo hace fácil de entender para todos, independientemente de tu experiencia en gestión de proyectos.
* **Versatilidad:** Además de representar el cronograma de tareas, se puede usar para visualizar la utilización de recursos, mostrando quién está asignado a cada tarea y cuándo, facilitando la gestión de recursos humanos y materiales.
* **Seguimiento del progreso:** Permite a los gestores de proyectos monitorear el avance de las tareas a lo largo del tiempo, ayudando a identificar retrasos y ajustar el cronograma según sea necesario.

**Diagrama de Precedencias o Actividades**

Es una herramienta de gestión de proyectos que representa gráficamente las actividades y sus dependencias. Cada actividad tiene una duración definida, precedentes y un producto o entregable. Los hitos son puntos en el tiempo que marcan eventos importantes, como el inicio o fin de una actividad, y no tienen duración.

El modelo de precedencias define relaciones entre actividades y permite calcular la duración total del proyecto.

El diagrama de precedencias es un modelo simplificador, con las siguientes características principales:

* **Duraciones fijas:** Cada actividad tiene una duración establecida, facilitando la planificación.
* **Limitaciones de inicio:** No permite indicar el comienzo de una actividad en un punto específico de la ejecución de otra. Para solucionar esto:
  + **Dividir actividades:** Las actividades complejas se pueden segmentar.
  + **Uso de herramientas:** Por ejemplo, se puede establecer que una actividad no comience hasta que su actividad precedente esté cumplida en un 90 %.
* **Grafo ordenado totalmente:** Las actividades y sus relaciones están organizadas en un grafo que muestra la secuencia de tareas.
* **Representación de tareas:** Las tareas se representan como nodos en el diagrama.
* **Relaciones entre tareas:** Las flechas que conectan los nodos representan las relaciones de precedencia, indicando el flujo de una tarea a otra.

Todos los nodos tienen el mismo tamaño y pueden contener mucha información sobre la tarea. La estructura de la actividad en el grafo contiene:

* **Descripción de la actividad:** nombre dado a la actividad.
* **Etiqueta actividad:** número que identifica a cada actividad.
* **Duración:** tiempo que se tardará en completar la tarea.
* **Inicio temprano:** fecha más temprana en que puede comenzar la tarea.
* **Final temprano:** fecha más temprana en que puede finalizar la tarea.
* **Inicio tardío:** fecha más retrasada en la que se puede comenzar la tarea sin que afecte a la fecha final del proyecto.
* **Final tardío:** fecha más retrasada en la que puede terminar la tarea sin que afecte a la fecha final del proyecto.
* **Máximo tiempo disponible (MTD):** tiempo máximo que puede durar una tarea en caso de comenzar en su Inicio temprano y concluir en su Final tardío.
* **Holgura (H):** tiempo que disponemos para jugar con el inicio de la tarea, sin afectar al proyecto.
* **Camino Critico:** Es el conjunto de actividades o tareas que tienen holgura cero, cualquier retraso en estas actividades afectará directamente la duración total del proyecto.

**Análisis del Riesgo**

Tiene que ver con la identificación de los riesgos y los planes para minimizar sus efectos en el proyecto. Podemos clasificar los riesgos en tres categorías:

* **Riesgos del proyecto:** Identifican problemas de presupuesto, de agenda, del personal… Afectan al coste y duración del proyecto.
* **Riesgos del producto:** Problemas en el diseño, implementación, interfaz, incertidumbre técnica… Afectan a la calidad del software resultante.
* **Riesgos del negocio:** Cambio en la dirección, cambios de estrategia del negocio… Afectan al equipo de desarrollo y a la realización del proyecto en sí.

**Proceso de gestión del riesgo**

**Identificación del riesgo**

* Identifica los posibles riesgos del proyecto, del producto y del negocio.
* Determina para cada tipo de situación, los posibles riesgos que pueden afectar al desarrollo del proyecto.

**Estimación y evaluación del riesgo**

* Determina la probabilidad y el impacto de cada riesgo.
* La probabilidad puede ser expresada de forma cuantitativa o cualitativa: muy bajo (<10%), bajo (10-25%), moderado (25-50%), alto (50-75%), muy alto (>75 %).
* El impacto tiene que ver con sus consecuencias: catastrófica (amenazan la supervivencia del proyecto), grave (causarían grandes demoras), tolerable (demoras dentro de la contingencia permitida) o insignificante, y con su duración.

**Planificación del riesgo**

* Traza un plan para evitar o minimizar los riesgos.
* Considerar cada riesgo con probabilidad alta a partir de un impacto tolerable y moderada a partir de un impacto serio y desarrollar estrategias para gestionarlo.
* Se plantean escenarios “What-if”.
* Se diseñan estrategias para manejar el riesgo:
  + **Estrategias de evitación:** buscan reducir la probabilidad de que ocurra el riesgo.
  + **Estrategias de minimización:** buscan reducir el impacto del riesgo.
  + **Planes de contingencia:** son estrategias para gestionar el peor escenario posible.
* Todas estas tareas se engloban y detallan en el plan de gestión y supervisión del riesgo.

**Control del riesgo**

* Controla la ocurrencia de riesgos durante el proyecto.
* Asegura el cumplimiento de las tareas para evitar riesgos y minimizar su impacto.
* Revisa periódicamente los riesgos identificados para decidir si su probabilidad de ocurrencia ha aumentado o disminuido.
* Revisa si las consecuencias del riesgo cambian.
* Los riesgos considerados deben ser discutidos en las reuniones periódicas.

**Técnicas de Estimación**

Técnicas de Estimación en el desarrollo de software están diseñadas para estimar el tiempo y esfuerzo necesarios para completar un proyecto. Estas técnicas son fundamentales en la planificación del proyecto, ya que permiten una mejor organización de recursos y del tiempo. Se basan en datos históricos, a partir de métricas sobre el proceso que ofrecen una perspectiva sobre experiencias previas. Permitiendo que las estimaciones se basen en la realidad de proyectos anteriores, proporcionando mayor precisión en las previsiones.

La experiencia previa del equipo juega un papel importante en la estimación, conocer la dinámica del trabajo y las posibles dificultades ayuda a prever con mayor exactitud el tiempo y esfuerzo requeridos. Un equipo con experiencia puede identificar las tareas más complicadas y las áreas con mayor riesgo o que ocupen más tiempo. Obteniéndose una estimación más realista, beneficiando a la asignación de tareas y gestión de recursos.

El proceso de estimación requiere acceso a información fiable y el compromiso de todas las partes involucradas en el proyecto. Esto es esencial, ya que las estimaciones influyen en los plazos y costos. A pesar de los esfuerzos, las estimaciones siempre conllevan un nivel de riesgo e incertidumbre. Este está asociado a factores como la complejidad del proyecto, su tamaño y el grado de especificación de sus requisitos.

En metodologías evolutivas o incrementales, las estimaciones se pueden ajustar conforme cambian los requisitos. A medida que el desarrollo avanza, es posible revisar y modificar la planificación, ayudando a mitigar el riesgo asociado a cambios imprevistos. Aunque la estimación de costes/esfuerzo no es una ciencia exacta, existen pasos sistemáticos que pueden seguirse para lograr estimaciones más fiables y con menor riesgo.

Existen varios enfoques para realizar estimaciones en proyectos de software:

1. **Estimación tardía:** ofrece mayor fiabilidad, aunque puede resultar poco práctica debido al retraso en su obtención.
2. **Basada en proyectos anteriores**: no garantiza que se repitan las mismas condiciones de los proyectos pasados.
3. **Uso de técnicas de descomposición:** estrategia que sigue el principio de “divide y vencerás”, el proyecto se fragmenta en partes para estimar cada componente.
4. **Uso de métodos empíricos:** basado en la combinación de datos históricos para generar estimaciones más fundamentadas.

Cabe destacar que las técnicas de descomposición y los métodos empíricos no son excluyentes, sino complementarios.

Las estimaciones suelen basarse en diversas métricas, que se dividen en tres categorías principales:

* Sobre el proceso, consideran aspectos como metodologías y herramientas utilizadas.
* Sobre el proyecto, evalúan factores relacionados con el equipo.
* Sobre el producto, que abarcan características como el tamaño y la complejidad del software a desarrollar.

Las **métricas de software** son herramientas útiles para evaluar de forma cuantitativa diversos aspectos del desarrollo del proyecto. Su objetivo es generar indicadores que ayuden a mejorar del rendimiento en el proceso o ajustar el flujo de trabajo. Es importante interpretarlas con precaución, ya que ofrecen solo una visión parcial de la realidad. No existe una única métrica universalmente valida, por lo que hay que relativizar según el contexto, como el lenguaje de programación. Estas métricas no deben emplearse para criticar a individuos o equipos.

La clasificación histórica de las métricas de software se organiza en diferentes categorías:

* **Orientadas al tamaño:** miden errores o defectos por líneas de código (LOC).
* **Orientadas a la función:** como los puntos de función (FP).
* **Orientadas a casos de uso:** se relacionan con el tamaño del sistema y las pruebas.
* **Para sistemas orientados a objetos:** incluyen factores como la herencia, cohesión y acoplamiento.

Las **técnicas de estimación** basadas en **descomposición** consiste en dividir el proyecto en partes más pequeñas para facilitar la estimación. Puede realizarse a nivel del problema, dividiendo el sistema en módulos o a nivel del proceso, separando las tareas involucradas. La precisión de estas técnicas depende de una correcta estimación del tamaño del proyecto. El valor final se calcula a partir de tres escenarios: optimista, más probable (con mayor peso) y pesimista.

Los **métodos de estimación empíricos** se basan en fórmulas que predicen el esfuerzo a partir de mediciones como líneas de código (LOC) o puntos de función (FP), utilizando técnicas de regresión. Toman como datos de entrada una muestra de proyectos finalizados para generar sus predicciones. Puede requerir recalibración al compararse con resultados reales. Un ejemplo son los modelos de estimación de costes COCOMO, que ofrecen una jerarquía de modelos con distintos fines.

Escenarios alternativos en la planificación del desarrollo de software incluyen diversas opciones estratégicas:

* **Adquisición vs. Desarrollo de software:** Esta decisión depende de la criticidad del desarrollo y los costes implicados. Se analizan distintas alternativas y se estima la probabilidad de cada escenario. El coste total se calcula multiplicando la probabilidad por el coste de cada alternativa.
* **Externalización del desarrollo (outsourcing):** Implica contratar parte del desarrollo a una empresa externa para reducir costes y mejorar la calidad. La decisión puede ser de carácter estratégico, para reducir costes, o táctica, al recurrir a una empresa especializada. Requiere un análisis financiero, y aunque permite reducir la necesidad de recursos internos, se pierde control sobre el proceso.